`dotnet_pkg_info` Tool

1.1. Introduction

Microsoft *.NET core* is a cross-platform framework that can be used to build ASP.NET Core web applications, command-line applications, libraries, and universal windows platform applications. The .NET applications are written in *C#, F#, or Visual Basic* programming languages. .NET applications mainly use MSBuild as the build system. The *MSBuild* files, commonly referred as *project files* adhere to MSBuild XML schema. These *project files* end with extensions `.csproj`, `.vbproj` or `.fsproj`. .NET packages that use *Microsoft Visual Studios* for development have a Visual Studios Solution file (`.sln` extension) in the package root directories. A visual studios *solution* files for a package is a text file that has information such as *project* file paths and *build configurations* (`Debug` or `Release`) for the modules in the project.

Given a .NET package, `dotnet_pkg_info` is a *command line* tool that obtains information relevant to SWAMP about the package. The tool is intended to be used by SWAMP UI team to help users add .NET packages to SWAMP. Code from `dotnet_pkg_info` tool will also be used in the SWAMP backend to get information about the package during the build.

List of functionality of `dotnet_pkg_info`

1. List *Visual Studios Solution Files*, *Project Files*, *Target Frameworks* and *Build Configuration* in a given package
2. Show common .NET *File Type Extensions*
3. Show *Target Frameworks* on a SWAMP platform

1.2. List *Visual Studios Solution Files*, *Project Files*, *Target Frameworks* and *Build Configuration* in a given package.

Given the path to an unarchived package as an argument, The `dotnet_pkg_info` tool, starting in the project root directory (this is selected by the user), recursively looks for files with `.sln` extension . The files with `.sln` extension are Visual Studios Solution Files. These solution files are text based files that contain paths to Dotnet Project Files along with information such as build configurations (`Debug`, `Release`) for each of the modules. The argument can also be the path to a *solution* file or a *project* file.

Table 1. Valid Target Framework and Target Framework Moniker:

Target Framework	Target Framework Moniker	Windows Only
.NET Standard	netstandard1.0 netstandard1.1 netstandard1.2 netstandard1.3 netstandard1.4 netstandard1.5 netstandard1.6 netstandard2.0	False

Target Framework	Target Framework Moniker	Windows Only
.NET Core	netcoreapp1.0 netcoreapp1.1 netcoreapp2.0 netcoreapp2.1	False
.NET Framework	net11 net20 net35 net40 net403 net45 net451 net452 net46 net461 net462 net47 net471 net472	True
Windows Store	netcore [netcore45] netcore45 [win] [win8] netcore451 [win81]	True
.NET Micro Framework	netmf	True
Silverlight	sl4 sl5	True
Windows Phone	wp [wp7] wp7 wp75 wp8 wp81 wpa81	True
Universal Windows Platform	uap [uap10.0] uap10.0 [win10] [netcore50]	False

This list may be change, refer to [https://docs.com/en-us/dotnet/standard/frameworks] for the update list.

To get SWAMP related information about a package, execute `dotnet_pkg_info` with `--package` option, along with the path to a package directory or a solution file or a project file. If the argument is a package directory, `dotnet_pkg_info` recursively searches for *solution* files in the package. For each of the *solution files*, it lists the *project files*. And for each of the *project files*, `dotnet_pkg_info` list the given *target frameworks* and *build configurations*. The options and arguments for `dotnet_pkg_info` are listed in Table-2

Table 2. dotnet_pkg_info Options and Arguments:

Option	Description
`--package PACKAGE`	Path to the package directory or a solution file or a project file
`--format FORMAT`	the values `text` or `json`. Default is `json`

Option	Description
`--no-config`	Do not display configuration information
`--no-framework`	Do not display target framework information
`--src-file-types`	list of .NET source file extensions
`--framework-types`	list of frameworks available on
`--proj-file-types`	list of .NET msbuild project file extensions

Example

```
% dotnet_pkg_info --package './Identity-2.0.1'
```

Output

```
{
  "sln_files": {
    "Identity.sln": [
      "src/AspNetCore.Identity/AspNetCore.Identity.csproj",
      "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj"
    ],
    "IdentityCore.sln": [
      "src/AspNetCore.Identity/AspNetCore.Identity.csproj",
      "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj",
      "src/Extensions.Identity.Stores/Extensions.Identity.Stores.csproj"
    ]
  },
  "proj_files": {
    "src/AspNetCore.Identity/AspNetCore.Identity.csproj": {
      "frameworks": ["netcoreapp2.0", "net461"],
      "configuration": ["Debug", "Release"],
      "default_framework": "netcoreapp2.0",
      "default_configuration": "Debug"
    },
    "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj": {
      "frameworks": ["netstandard2.0"],
      "configuration": ["Debug", "Release"],
      "default_framework": "netstandard2.0",
      "default_configuration": "Debug"
    },
    "src/Extensions.Identity.Stores/Extensions.Identity.Stores.csproj": {
      "frameworks": ["netstandard2.0"],
      "configuration": ["Debug", "Release"],
      "default_framework": "netstandard2.0",
      "default_configuration": "Debug"
    }
  }
}
```

Example with Text Output:

```
% dotnet_pkg_info --format text --package './Identity-2.0.1'
```

Output

```
sln_files:
  Identity.sln
    src/AspNetCore.Identity/AspNetCore.Identity.csproj
    src/Extensions.Identity.Core/Extensions.Identity.Core.csproj
  IdentityCore.sln
    src/AspNetCore.Identity/AspNetCore.Identity.csproj
    src/Extensions.Identity.Core/Extensions.Identity.Core.csproj
    src/Extensions.Identity.Stores/Extensions.Identity.Stores.csproj
proj_files:
  src/AspNetCore.Identity/AspNetCore.Identity.csproj
    frameworks:
      netcoreapp2.0
      net461
    configuration:
      Debug
      Release
    default_framework:
      netcoreapp2.0
    default_configuration:
      Debug
  src/Extensions.Identity.Core/Extensions.Identity.Core.csproj
    frameworks:
      netstandard2.0
    configuration:
      Debug
      Release
    default_framework:
      netstandard2.0
    default_configuration:
      Debug
  src/Extensions.Identity.Stores/Extensions.Identity.Stores.csproj
    frameworks:
      netstandard2.0
    configuration:
      Debug
      Release
    default_framework:
      netstandard2.0
    default_configuration:
      Debug
```

NOTE To get package information without *Build Configuration* and *Target Framework* information, use `--no-config` and `--no-framework` option to the `dotnet_pkg_info` command.

1.2.1. For packages without solution files

If a package does not have a *solution file* in the package root directory, the tool recursively searches the package for *project files*. It lists the *project files* along with *target frameworks* mentioned in the *project files*. Note that *build configuration* information won't be available in this case as *build configuration* is provided in the *solution files*.

1.3. Target Frameworks on SWAMP platforms

To display *target frameworks* available on a SWAMP platform, use '--framework-types' option with `dotnet_pkg_info` tool.

Example

```
dotnet_pkg_info --framework-types
```

Output

```
{
  ".NET Standard": {
      "tf_moniker" : [
          "netstandard1.0",
          "netstandard1.1",
          "netstandard1.2",
          "netstandard1.3",
          "netstandard1.4",
          "netstandard1.5",
          "netstandard1.6",
          "netstandard2.0",
          "netcoreapp1.0",
          "netcoreapp1.1",
          "netcoreapp2.0",
          "netcoreapp2.1"
      ],
      "windows_only": false
   },
  ".NET Core" : {
     "tf_moniker" : [
         "netcoreapp1.0",
         "netcoreapp1.1",
         "netcoreapp2.0",
         "netcoreapp2.1"
     ],
     "windows_only": false
   },
   ".NET Framework" : {
     "tf_moniker" : [
        "net11",
        "net20",
```

```
        "net35",
        "net40",
        "net403",
        "net45",
        "net451",
        "net452",
        "net46",
        "net461",
        "net462",
        "net47",
        "net471",
        "net472"
      ],
      "windows_only": true
    },
    "Windows Store": {
      "tf_moniker" : [
        "netcore [netcore45]",
        "netcore45 [win] [win8]",
        "netcore451 [win81]"
      ],
      "windows_only": true
    },
    ".NET Micro Framework": {
      "tf_moniker" : [
        "netmf"
      ],
      "windows_only": true
    },
    "Silverlight": {
      "tf_moniker" : [
        "sl4",
        "sl5"
      ],
      "windows_only": true
    },
    "Windows Phone": {
      "tf_moniker" : [
        "wp [wp7]",
        "wp7",
        "wp75",
        "wp8",
        "wp81",
        "wpa81"
      ],
      "windows_only": true
    },
    "Universal Windows Platform": {
      "tf_moniker" : [
        "uap",
        "uap10.0"
```

```
      ],
      "windows_only": false
    }
}
```

1.4. Show .NET File Extensions

Lists the .NET file types extensions

Example

```
% dotnet_pkg_info --src-file-types
```

Output

```
{
  ".cs": {
    "description": "C# source files",
    "windows_only": false
  },
  ".vb": {
    "description": "Visual Basics source files",
    "windows_only": true
  },
  ".fs": {
    "description": "F# source files",
    "windows_only": true
  }
}
```

1.5. Show .NET Project File Extensions

Lists the .NET project file extensions

```
% dotnet_pkg_info --project-file-types
```

Output

```
{
  ".csproj": {
    "description": "csharp project file"
  },
  ".vbproj": {
    "description": "Visual Basics project files"
  },
  ".fsproj": {
  "description": "fsharp project file"
  }
}
```

1.6. Package info to the backend

If a user selects a *solution* file, and a certain set of *project* files and *target* frameworks and *build* configuration for their package. The SWAMP UI or middleware should pass the .NET package information to the backend in a `json` format. The information in the `json` format must be assigned to the `package-dotnet-info` attribute in the `package.conf` file.

The format for the `package-dotnet-info` should be same as the `json` output produced by `dotnet_pkg_info` tool, except for the values for `framework` and `configuration` attributes should be a single string value and not a list of string, and attributes `default_framework` and `default_configuration` should not be present.

Example:

```
{
  "sln_files": {
    "Identity.sln": [
      "src/AspNetCore.Identity/AspNetCore.Identity.csproj",
      "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj"
    ],
  },
  "proj_files": {
    "src/AspNetCore.Identity/AspNetCore.Identity.csproj": {
      "framework": "netcoreapp2.0",
      "configuration": "Debug",
    },
    "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj": {
      "framework": "netstandard2.0",
      "configuration": "Debug"
    },
  }
}
```

Scenario 1:

User selects a *solution* but does not select projects and does not configure the projects. In this case, `package-dotnet-info` can list the *solution* file with empty list for projects. The *assessment framework*

invokes the MSBuild system with the *solution* file as the argument. i.e. all the modules in the *solution* file will be built against frameworks and configuration provided in the *project* files for all the modules.

Example:

```
{
  "sln_files": {
    "Identity.sln": []
  }
}
```

Scenario 2:

User selects a *solution*, and one or more projects in the *solution*, but does not select *configuration* for the projects. In this case, `package-dotnet-info` can list the *solution* file with the list of projects selected by the user. The *assessment framework* invokes the MSBuild system for each of the selected projects. The projects will be built against frameworks and configuration provided in the selected *project* files.

Example:

```
{
  "sln_files": {
    "Identity.sln": [
      "src/AspNetCore.Identity/AspNetCore.Identity.csproj",
      "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj"
    ]
  },
  "proj_files": {
    "src/AspNetCore.Identity/AspNetCore.Identity.csproj": {
    },
    "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj": {
    },
  }
}
```

Scenario 2:

User selects a *solution*, and one or more projects in the *solution*, but also selects *frameworks* and *configuration* for the projects. The *assessment framework* invokes the MSBuild system for each of the selected projects. The projects will be built against frameworks and configuration provided by the user.

Example:

```
{
    "src/AspNetCore.Identity/AspNetCore.Identity.csproj": {
      "framework": "netcoreapp2.0",
      "configuration": "Debug"
    },
    "src/Extensions.Identity.Core/Extensions.Identity.Core.csproj": {
      "framework": "netstandard2.0",
      "configuration": "Debug"
    },
    "src/Extensions.Identity.Stores/Extensions.Identity.Stores.csproj": {
      "framework": "netstandard2.0",
      "configuration": "Release"
    }
}
```